

E13 BP Algorithm (C++/Python)

16337102 Zilin Huang

December 6, 2018

Contents

1	Horse Colic Data Set	2
2	Reference Materials	2
3	Tasks	7
4	Codes and Results	7
4.1	Codes	7
4.2	Result	14

1 Horse Colic Data Set

The description of the horse colic data set (<http://archive.ics.uci.edu/ml/datasets/Horse+Colic>) is as follows:

We aim at trying to predict if a horse with colic will live or die.

Note that we should deal with missing values in the data! Here are some options:

- Use the features mean value from all the available data.
- Fill in the unknown with a special value like -1.
- Ignore the instance.
- Use a mean value from similar items.
- Use another machine learning algorithm to predict the value.

2 Reference Materials

1. Stanford: **CS231n: Convolutional Neural Networks for Visual Recognition** by Fei-Fei Li, etc.

- Course website: <http://cs231n.stanford.edu/2017/syllabus.html>
- Video website: https://www.bilibili.com/video/av17204303/?p=9&tdsourcetag=s_pctim_aiomsg

2. **Machine Learning** by Hung-yi Lee

- Course website: <http://speech.ee.ntu.edu.tw/~tlkagk/index.html>
- Video website: <https://www.bilibili.com/video/av9770302/from=search>

3. A Simple neural network code template

```
1 # -*- coding: utf-8 -*-
2 import random
3 import math
4
5 # Shorthand:
6 # "pd_" as a variable prefix means "partial derivative"
7 # "d_" as a variable prefix means "derivative"
8 # "_wrt_" is shorthand for "with respect to"
9 # "w_ho" and "w_ih" are the index of weights from hidden to output layer neurons
   and input to hidden layer neurons respectively
10
11 class NeuralNetwork:
```

```

12 LEARNING_RATE = 0.5
13 def __init__(self, num_inputs, num_hidden, num_outputs, hidden_layer_weights
    = None, hidden_layer_bias = None, output_layer_weights = None,
    output_layer_bias = None):
14     #Your Code Here
15
16 def init_weights_from_inputs_to_hidden_layer_neurons(self,
    hidden_layer_weights):
17     #Your Code Here
18
19 def init_weights_from_hidden_layer_neurons_to_output_layer_neurons(self,
    output_layer_weights):
20     #Your Code Here
21
22 def inspect(self):
23     print('_____')
24     print('*_Inputs:_{ }'.format(self.num_inputs))
25     print('_____')
26     print('Hidden_Layer')
27     self.hidden_layer.inspect()
28     print('_____')
29     print('*_Output_Layer')
30     self.output_layer.inspect()
31     print('_____')
32
33 def feed_forward(self, inputs):
34     #Your Code Here
35
36 # Uses online learning, ie updating the weights after each training case
37 def train(self, training_inputs, training_outputs):
38     self.feed_forward(training_inputs)
39
40     # 1. Output neuron deltas
41     #Your Code Here
42     # E / z
43
44     # 2. Hidden neuron deltas
45     # We need to calculate the derivative of the error with respect to the
46     output of each hidden layer neuron
47     # dE/dy = E / z * z / y = E / z * w
48     # E / z = dE/dy * z /

```

```

48     #Your Code Here
49
50     # 3. Update output neuron weights
51     
$$\# \quad E / w = E / z * z / w$$

52     
$$\# \quad w = * E / w$$

53     #Your Code Here
54
55     # 4. Update hidden neuron weights
56     
$$\# \quad E / w = E / z * z / w$$

57     
$$\# \quad w = * E / w$$

58     #Your Code Here
59
60     def calculate_total_error(self, training_sets):
61         #Your Code Here
62         return total_error
63
64 class NeuronLayer:
65     def __init__(self, num_neurons, bias):
66
67         # Every neuron in a layer shares the same bias
68         self.bias = bias if bias else random.random()
69
70         self.neurons = []
71         for i in range(num_neurons):
72             self.neurons.append(Neuron(self.bias))
73
74     def inspect(self):
75         print('Neurons:', len(self.neurons))
76         for n in range(len(self.neurons)):
77             print('└Neuron', n)
78             for w in range(len(self.neurons[n].weights)):
79                 print('└└Weight:', self.neurons[n].weights[w])
80             print('└└Bias:', self.bias)
81
82     def feed_forward(self, inputs):
83         outputs = []
84         for neuron in self.neurons:
85             outputs.append(neuron.calculate_output(inputs))
86         return outputs
87
88     def get_outputs(self):

```

```

89         outputs = []
90         for neuron in self.neurons:
91             outputs.append(neuron.output)
92         return outputs
93
94 class Neuron:
95     def __init__(self, bias):
96         self.bias = bias
97         self.weights = []
98
99     def calculate_output(self, inputs):
100         #Your Code Here
101
102     def calculate_total_net_input(self):
103         #Your Code Here
104
105     # Apply the logistic function to squash the output of the neuron
106     # The result is sometimes referred to as 'net' [2] or 'net' [1]
107     def squash(self, total_net_input):
108         #Your Code Here
109
110     # Determine how much the neuron's total input has to change to move closer to
111     # the expected output
112     #
113     # Now that we have the partial derivative of the error with respect to the
114     # output (  $E / y$  ) and
115     # the derivative of the output with respect to the total net input (  $dy / dz$  ) we can calculate
116     # the partial derivative of the error with respect to the total net input.
117     # This value is also known as the delta ( ) [1]
118     #  $\delta = E / z = E / y * dy / dz$ 
119     #
120     def calculate_pd_error_wrt_total_net_input(self, target_output):
121         #Your Code Here
122
123     # The error for each neuron is calculated by the Mean Square Error method:
124     def calculate_error(self, target_output):
125         #Your Code Here
126
127     # The partial derivate of the error with respect to actual output then is
128     calculated by:

```

```

126 # = 2 * 0.5 * (target output - actual output) ^ (2 - 1) * -1
127 # = -(target output - actual output)
128 #
129 # The Wikipedia article on backpropagation [1] simplifies to the following,
    but most other learning material does not [2]
130 # = actual output - target output
131 #
132 # Alternative, you can use (target - output), but then need to add it during
    backpropagation [3]
133 #
134 # Note that the actual output of the output neuron is often written as y
    and target output as t so:
135 # = E / y = -(t - y)
136 def calculate_pd_error_wrt_output(self, target_output):
137     #Your Code Here
138
139     # The total net input into the neuron is squashed using logistic function to
    calculate the neuron's output:
140     # y = 1 / (1 + e^(-z))
141     # Note that where represents the output of the neurons in whatever layer
    we're looking at and represents the layer below it
142     #
143     # The derivative (not partial derivative since there is only one variable) of
    the output then is:
144     # dy / dz = y * (1 - y)
145     def calculate_pd_total_net_input_wrt_input(self):
146         #Your Code Here
147
148     # The total net input is the weighted sum of all the inputs to the neuron and
    their respective weights:
149     # z = net = x w + x w ...
150     #
151     # The partial derivative of the total net input with respect to a given
    weight (with everything else held constant) then is:
152     # = z / w = some constant + 1 * x w ^ (1-0) + some constant ...
    = x
153     def calculate_pd_total_net_input_wrt_weight(self, index):
154         #Your Code Here
155
156     # An example:
157

```

```

158 nn = NeuralNetwork(2, 2, 2, hidden_layer_weights=[0.15, 0.2, 0.25, 0.3],
    hidden_layer_bias=0.35, output_layer_weights=[0.4, 0.45, 0.5, 0.55],
    output_layer_bias=0.6)
159 for i in range(10000):
160     nn.train([0.05, 0.1], [0.01, 0.99])
161     print(i, round(nn.calculate_total_error([[[0.05, 0.1], [0.01, 0.99]]]), 9))

```

3 Tasks

- Given the training set `horse-colic.data` and the testing set `horse-colic.test`, implement the BP algorithm and establish a neural network to predict if horses with colic will live or die. In addition, you should calculate the accuracy rate.
- Please submit a file named `E13_YourNumber.pdf` and send it to `ai_2018@foxmail.com`

4 Codes and Results

4.1 Codes

exp.py

```

1 import random
2 import math
3 import numpy as np
4 import pandas as pd
5 import copy
6
7 class NeuralNetwork:
8     LEARNING_RATE = 0.5
9
10     def __init__(self, num_inputs, num_hidden, num_outputs, hidden_layer_weights=None,
        hidden_layer_bias=None, output_layer_weights=None, output_layer_bias=None):
11         #Your Code Here
12         self.num_in = num_inputs
13         self.num_hid = num_hidden
14         self.num_out = num_outputs
15         self.hidden_layer = NeuronLayer(num_hidden, hidden_layer_bias)
16         self.output_layer = NeuronLayer(num_outputs, output_layer_bias)
17         self.init_weights_from_inputs_to_hidden_layer_neurons(hidden_layer_weights)

```

```

18         self.init_weights_from_hidden_layer_neurons_to_output_layer_neurons(
19             output_layer_weights)
20
21 #
22 def init_weights_from_inputs_to_hidden_layer_neurons(self, hidden_layer_weights):
23     if hidden_layer_weights:
24         counter = 0
25         for i in range(self.num_hid):
26             self.hidden_layer.neurons[i].weights = hidden_layer_weights[counter:
27                 counter + self.num_in - 1]
28             counter += self.num_in
29     else:
30         for i in range(self.num_hid):
31             weights = [random.random() for i in range(self.num_in)]
32             self.hidden_layer.neurons[i].weights = copy.deepcopy(weights)
33
34 def init_weights_from_hidden_layer_neurons_to_output_layer_neurons(self,
35     output_layer_weights):
36     if output_layer_weights:
37         counter = 0
38         for i in range(self.num_out):
39             self.output_layer.neurons[i].weights = output_layer_weights[counter:
40                 counter + self.num_hid - 1]
41             counter += self.num_hid
42     else:
43         for i in range(self.num_out):
44             weights = [random.random() for i in range(self.num_hid)]
45             self.output_layer.neurons[i].weights = copy.deepcopy(weights)
46
47 def inspect(self):
48     print('————')
49     print('*_Inputs:_{ }'.format(self.num_in))
50     print('————')
51     print('Hidden_Layer')
52     self.hidden_layer.inspect()
53     print('————')
54     print('*_Output_Layer')
55     self.output_layer.inspect()
56     print('————')
57
58 def feed_forward(self, inputs):

```



```

55     mids = self.hidden_layer.feed_forward(inputs)
56     self.output_layer.feed_forward(mids)
57     total = 0
58     for i in range(self.num_out):
59         total += self.output_layer.neurons[i].output
60     for i in range(self.num_out):
61         self.output_layer.neurons[i].output /= total
62
63     def train(self, training_inputs, training_outputs):
64         self.feed_forward(training_inputs)
65
66         deltas_out = []
67         for i in range(self.num_out):
68             delta = self.hidden_layer.neurons[i].
69                 calculate_pd_error_wrt_total_net_input(int(training_outputs[i]))
70             deltas_out.append(delta)
71
72         deltas_hid = []
73         for i in range(self.num_hid):
74             dE_dyi = 0
75             for j in range(self.num_out):
76                 dE_dyi += self.output_layer.neurons[j].weights[i] * deltas_out[j]
77             a = self.hidden_layer.neurons[i].output
78             delta = a * (1 - a) * dE_dyi
79             deltas_hid.append(delta)
80
81         w_out = [ [] for i in range(self.num_out) ]
82         for i in range(self.num_out):
83             for j in range(self.num_hid):
84                 w = self.LEARNING_RATE * \
85                     deltas_out[i] * \
86                     self.hidden_layer.neurons[j].output
87                 w_out[i].append(w)
88         for i in range(self.num_out):
89             for j in range(self.num_hid):
90                 self.output_layer.neurons[i].weights[j] += w_out[i][j]
91
92         w_hid = [ [] for i in range(self.num_hid) ]
93         for i in range(self.num_hid):
94             for j in range(self.num_in):
95                 w = self.LEARNING_RATE * \

```

```

95         deltas_hid[i] * \
96         self.hidden_layer.neurons[i].
           calculate_pd_total_net_input_wrt_weight(
97             j)
98         w_hid[i].append(w)
99     for i in range(self.num_hid):
100         for j in range(self.num_hid):
101             self.hidden_layer.neurons[i].weights[j] += w_hid[i][j]
102
103
104 class NeuronLayer:
105     def __init__(self, num_neurons, bias):
106
107         # Every neuron in a layer shares the same bias
108         self.bias = bias if bias else random.random()
109
110         self.neurons = []
111         for i in range(num_neurons):
112             self.neurons.append(Neuron(self.bias))
113
114     def inspect(self):
115         print('Neurons:', len(self.neurons))
116         for n in range(len(self.neurons)):
117             print('  _Neuron', n)
118             for w in range(len(self.neurons[n].weights)):
119                 print('    _Weight:', self.neurons[n].weights[w])
120             print('    _Bias:', self.bias)
121
122     def feed_forward(self, inputs):
123         outputs = []
124         for neuron in self.neurons:
125             outputs.append(neuron.calculate_output(inputs))
126         return outputs
127
128     def get_outputs(self):
129         outputs = []
130         for neuron in self.neurons:
131             outputs.append(neuron.output)
132         return outputs
133
134 class Neuron:

```

```

135     def __init__(self, bias):
136         self.bias = bias
137         self.weights = []
138         self.weighted_input = 0
139         self.output = 0
140         self.inputs = []
141
142     #
143     def calculate_output(self, inputs):
144         self.weighted_input = self.calculate_total_net_input(inputs)
145         output = self.squash(self.weighted_input)
146         self.output = output
147         return output
148
149     def calculate_total_net_input(self, inputs):
150         total_net_input = 0
151         self.inputs = copy.deepcopy(inputs)
152         num = len(self.weights)
153         for i in range(num):
154             total_net_input += self.weights[i] * inputs[i]
155         total_net_input += self.bias
156         return total_net_input
157
158     def squash(self, total_net_input):
159         down = 1 + pow(np.e, -total_net_input)
160         output = 1 / down
161         return output
162
163     def calculate_pd_error_wrt_total_net_input(self, target_output):
164         delta = self.calculate_pd_error_wrt_output(
165             target_output) * self.calculate_pd_total_net_input_wrt_input()
166         return delta
167
168     def calculate_error(self, target_output):
169         #Your Code Here
170         error = pow(self.output - target_output, 2) * 0.5
171         return error
172
173     def calculate_pd_error_wrt_output(self, target_output):
174         pd_error = (target_output - self.output)
175         return pd_error

```

```

176
177     def calculate_pd_total_net_input_wrt_input(self):
178         pd = self.output * (1 - self.output)
179         return pd
180
181     x
182
183     def calculate_pd_total_net_input_wrt_weight(self, index):
184         x_index = self.inputs[index]
185         return x_index
186
187
188 def test(network, data, result):
189     global result_dic
190     num = len(data)
191     correct = 0
192     outputs = []
193     for i in range(num):
194         inputs = data.iloc[i].tolist()
195         network.feed_forward(inputs)
196         max_output = 0
197         position = -1
198         for j in range(network.num_out):
199             if max_output < network.output_layer.neurons[j].output:
200                 max_output = network.output_layer.neurons[j].output
201                 position = j
202             outputs.append(result_dic[position])
203     for i in range(num):
204         if outputs[i] == result.iloc[i]:
205             correct += 1
206     return correct / num
207
208
209 result_dic = {0: '100', 1: '010', 2: '001'}
210 data = pd.read_csv('train.data', header=None)
211 result = data[22]
212 result = result.replace(
213     {-1: '100', 1: '100', 2: '010', 3: '001'})
214 data = data / 10
215
216 #
217 del data[2]
218 del data[22]
219 del data[25]
220 del data[26]
221 del data[27]

```

```

217
218 test_data = pd.read_csv('train.data', header=None)
219 test_result = test_data[22]
220 test_result = test_result.replace(
221     {-1: '100', 1: '100', 2: '010', 3: '001'})
222 test_data = test_data / 10
223 del test_data[2]
224 del test_data[22]
225 del test_data[25]
226 del test_data[26]
227 del test_data[27]
228
229 nn = NeuralNetwork(22, 6, 3, hidden_layer_weights=None,
230                    hidden_layer_bias=random.random(),
231                    output_layer_weights=None,
232                    output_layer_bias=random.random())
233 num = len(data)
234 for j in range(1, 121):
235     accuracy = 0
236     if j % 8 == 0:
237         print('after_training_' + str(j) + '_times, accuracy:')
238         accuracy = test(nn, test_data, test_result)
239         print(accuracy)
240         if accuracy >= 0.6:
241             nn.hidden_layer.inspect()
242             nn.output_layer.inspect()
243     for i in range(num):
244         nn.train(data.iloc[i].tolist(), result.iloc[i])

```

deal_data.py

```

1 def deal_data(input_name, output_name):
2     train_data = open(input_name)
3     f = open(output_name, 'w+')
4     data = []
5     while 1:
6         line_ = train_data.readline()
7         if not line_:
8             break
9         line_ = line_[:-1]
10        line_ = line_.split('_')
11        data.append(line_)

```

```

12 num = len(data)
13 for i in range(num):
14     attr_num = len(data[i])
15     no = False
16     for j in range(attr_num):
17         if data[i][j] == '?':
18             data[i][j] = '-1'
19         if j == 28:
20             no = True
21     if no:
22         data[i] = data[i][: -1]
23     print(','.join(data[i]), file=f)
24 train_data.close()
25 f.close()
26
27
28 deal_data('horse-colic.data', 'train.data')
29 deal_data('horse-colic.test', 'test.data')

```

4.2 Result

```

after training 8 times, accuracy:
0.2566666666666665

```

```

after training 120 times, accuracy:
0.5966666666666667

```